Dominik Wawszczak
numer indeksu: 440014
numer grupy: 1

Algorytmy Parametryzowane
praca domowa 1

Zadanie 1

Oznaczmy

$$V(G) \;=\; \{v_1, \ldots, v_n\} \quad \text{oraz} \quad S_n \;=\; \left\{\sigma \;:\; \sigma \in \{1, \ldots, n\}^{\{1,\ldots,n\}} \;\wedge\; \sigma \text{ jest bijekcją}\right\}.$$

Niech

$$\mathrm{cutwidth}_\sigma(i) \;=\; |\{(u,v) \;:\; u \in \{v_{\sigma_1} \ldots v_{\sigma_i}\} \;\wedge\; v \in \{v_{\sigma_{i+1}}, \ldots, v_{\sigma_n}\} \;\wedge\; (u,v) \in E(G)\}|,$$

gdzie $\sigma \in S_n$ jest dowolną permutacją. Celem jest znalezienie permutacji $\sigma \in S_n$, dla której wartość

$$\max_{i \in \{1,\ldots,n-1\}} \mathrm{cutwidth}_\sigma(i)$$

jest najmniejsza możliwa. Konkretnie, chcemy obliczyć tę wartość.

Zdefiniujmy funkcję

$$\mathrm{out}(X) \;=\; |\{(u,v) \;:\; u \in X \;\wedge\; v \in V(G) \setminus X \;\wedge\; (u,v) \in E(G)\}|,$$

gdzie X jest dowolnym podzbiorem $V(G)$. Wówczas

$$\mathrm{cutwidth}_\sigma(i) \;=\; \mathrm{out}(\{v_{\sigma_1}, \ldots, v_{\sigma_i}\}).$$

Oczywiście, dla konkretnego X, wartość $\mathrm{out}(X)$ można łatwo obliczyć w czasie $n^{\mathcal{O}(1)}$.

Pozwala nam to skorzystać z programowania dynamicznego po podzbiorach. Niech

$$\mathrm{dp}(X) \;=\; \min\left\{\max_{i \in \{1,\ldots,|X|-1\}} \mathrm{cutwidth}_\sigma(i) \;:\; \sigma \in S_n \;\wedge\; \{v_{\sigma_1}, \ldots, v_{\sigma_{|X|}}\} = X\right\},$$

gdzie X jest dowolnym podzbiorem $V(G)$. Wtedy

$$\begin{aligned} \mathrm{dp}(\emptyset) \;&=\; 0, \\ \mathrm{dp}(X) \;&=\; \min\{\max(\mathrm{dp}(X \setminus \{x\}), \mathrm{out}(X \setminus \{x\}) \;:\; x \in X\}. \end{aligned}$$

Naturalnie, wynikiem jest $\mathrm{dp}(\{1, \ldots, n\})$. Żeby obliczyć wartości dp dla wszystkich podzbiorów $V(G)$, można na przykład przeglądać je w kolejności niemalejących rozmiarów. Całkowita złożoność naszego algorytmu wynosi $2^n \cdot n^{\mathcal{O}(1)}$.

Zadanie 2

Lemat 1 Zbiór punktów S, w którym żadne trzy nie są współliniowe, nie tworzy wierzchołków wielokąta wypukłego wtedy i tylko wtedy, gdy istnieją punkty $A, B, C, D \in S$ takie, że punkt D leży wewnątrz $\triangle ABC$.

Dowód lematu 1 Implikacja „w lewo” jest oczywista, skupmy się więc na implikacji „w prawo”. Niech $\{H_1, \ldots, H_h\}$ będzie otoczką wypukłą zbioru S, przy czym zakładamy, że punkty te leżą na otoczce w tej kolejności zgodnie z kierunkiem przeciwnym do ruchu wskazówek zegara. Niech D będzie dowolnym punktem z S nieleżącym na otoczce oraz niech $A = H_1$. Istnieje wówczas dokładnie jedno $i \in \{2, \ldots, h-1\}$ takie, że punkty $H_2, \ldots, H_i$ leżą po jednej stronie prostej AD, a punkty $H_{i+1}, \ldots, H_h$ po drugiej. Biorąc $B = H_i$ oraz $C = H_{i+1}$ otrzymamy szukane punkty.

Z udowodnionego lematu wynika, że jeśli istnieją punkty $A, B, C, D \in S$ takie, że punkt D leży wewnątrz $\triangle ABC$, to co najmniej jeden z nich musi zostać usunięty z S. Obserwacja ta prowadzi do następującego algorytmu:

Algorithm 1 ConvexDeletion

procedure ConvexDeletion(S, k)
    if no four points $A, B, C, D \in S$ satisfy that D lies inside $\triangle ABC$ **then**
        return true
    end if
    if $k \leqslant 0$ **then**
        return false
    end if
    Choose points $(A, B, C, D) \in S$ such that D lies inside $\triangle ABC$
    return ConvexDeletion($S \setminus \{A\}, k-1$) **or** ConvexDeletion($S \setminus \{B\}, k-1$) **or** ConvexDeletion($S \setminus \{C\}, k-1$) **or** ConvexDeletion($S \setminus \{D\}, k-1$)
end procedure

Szukanie takich czwórek punktów A, B, C, D można łatwo zrealizować w czasie $\mathcal{O}(n^4)$, przeglądając wszystkie czwórki punktów, dla każdej licząc odpowiednie iloczyny wektorowe i porównując ich znaki. Można to również zrobić w czasie $\mathcal{O}(n \log n)$, obliczając otoczkę wypukłą przy pomocy algorytmu Grahama i korzystając z konstrukcyjnego dowodu lematu 1.

Głębokość drzewa rekurencji naszego algorytmu wynosi co najwyżej k, ponieważ przy wywołaniu rekurencyjnym parametr k zmniejsza się o 1. Każdy wierzchołek tego drzewa ma co najwyżej czterech synów, skąd dostajemy górne oszacowanie na liczbę wierzchołków drzewa:

$$\sum_{i=0}^{k} 4^i \;=\; \frac{4^{k+1}-1}{3} \;=\; \mathcal{O}\big(4^k\big).$$

Łączna złożoność naszego algorytmu wynosi zatem $\mathcal{O}\big(4^k\big) \cdot n^{\mathcal{O}(1)}$.